割り算回路設計，あの手この手

連載 4

漸近法を用いた除算回路

鈴木昌治

デバイスの記事

関連データ

今回は，乗算をベースにする漸近法を用いた除算回路の実現方法を解説する．具体的には，除算に特化した「反復乗算法」と汎用的な「Newton-Raphson法」の二つのアルゴリズムを用いた回路設計を紹介する．なお，本稿で設計した回路のソース・コードは本誌ホームページ（http://www.cqpub.co.jp/dwm/）からダウンロードできる．（編集部）

これまで3回にわたって，回復法や非回復法を用いた除算回路設計および高速化について話をしてきました．これらは，いずれも「被除数の中に除数がいくつ取れるか」という視点に立ったものでした．

今回は，「被除数と除数の比を求める」という視点で除算を実行する方法について考えてみましょう．代表例として，漸近法を取り上げます．

1 漸近法とは

たしか高校の数学だったと思いますが，漸化式というものを習ったと思います．初期値，およびi番目と$i+1$番目の変数の関係を定義することで，すべての変数を定義できるというものです．$i \to \infty$としたときに収束するか，なども議論されていたように記憶しています．

漸近法はこの考えかたを応用したものです．初期値からはじめてi番目から$i+1$番目の変数を求めることを繰り返し，特定の値に収束させることで，関数の近似を行うというものです（もちろん，無限にサイクルを繰り返すのは現実的ではないため，有限回のサイクル後に必要な精度を得るものとする）．漸近法の中には，例えばNewton-Raphson法（詳細は後述）に代表される汎用性の高いアルゴリズムもありますが，適用する関数によってくふうが凝らされたものもあります．まずは，除算に特化したアルゴリズムを紹介しましょう．

● 回復法も非回復法も実は漸近法の一種

ところで，回復法を用いた除算回路では次の式を繰り返し演算していたことを思い出してください．

$$\left.\begin{array}{l} pq_i = 1 \\ pr_{i+1} = pr_i - d \end{array}\right\} \quad (pr_i \geqq d \text{の場合}) \cdots\cdots (1\text{-}1)$$

$$\left.\begin{array}{l} pq_i = 0 \\ pr_{i+1} = pr_i \end{array}\right\} \quad (pr_i < d \text{の場合}) \cdots\cdots (1\text{-}2)$$

ここで，pr_i：部分余，pq_i：部分商，d：除数

実は，条件分岐があるとはいえ，回復法や非回復法なども漸近法の一種なのです．けた単位での漸近を反復することで，商を必要とする精度に追い込んでいきます．しかし，手段に筆算のイメージが強いためか，あるいは結果が商と余で誤差なしで与えられるためか，漸近法として語られることはないようです．

● 乗算ベースの漸近法

回復法や非回復法は，けた単位で加算（減算）を行う方法でした．つまり，サイクル当たり1ビットの商を求めようとすれば，kビットの結果を求めるのにkサイクル必要となるのです．

これに対し，今回は乗算をベースにすることで指数的に精度を上げられる方法を考えてみましょう．ここでは，「反復乗算法（あるいは，Goldschmidtのアルゴリズム）」と言われる方法を用いることで，反復するサイクルを$\log_2 k$に短縮できることを説明します．

商をq，被除数をz，除数をdとすると，$q = z/d$となり

ます．つまり，求めるものはzとdの比となり，式(2)の表現を用います．

$$q=\frac{z}{d}=\frac{z}{d}\cdot\frac{\prod_{i=0}^{k-1}x_i}{\prod_{i=0}^{k-1}x_i} \quad \cdots\cdots(2)$$

まず，この式が何を表しているかを説明します．これは，「iサイクル目でx_iという変数を分母と分子それぞれに乗算する」という操作を反復するものと考えてください．分母と分子に同一のものを乗算しているので，qとしては変化しないことに着目しましょう．ここで，iサイクル目における分母をd_i，分子をz_iとすると，式(3)が成立します．

$$\left.\begin{array}{l} d_{i+1}=d_i\cdot x_i \\ z_{i+1}=z_i\cdot x_i \end{array}\right\} \quad \cdots\cdots(3)$$

これが漸化式となり，初期値はdとzです．x_iを乗算する操作を反復することで$d_i\to 1$という漸近操作ができれば，$z_i\to q$という結果を得ることになります．つまり，分母を1に追い込んでいくことで，自動的に分子がqに追い込まれるというわけです．

では，x_iをどのように定義するかを説明しましょう．まず，$d_i\to 1$を実現することは$1-d_i\to 0$を実現することと等価と考えることができます．式(2)の分子の初期値は，

$$1/2\leqq d<1 \quad \cdots\cdots(4)$$

に正規化しておきます．すると，

$$0<1-d\leqq 1/2 \quad \cdots\cdots(5)$$

となります．このため，$1-d$のべき乗であれば0に近づけることが可能であると判断できます．ここでは乗算を用いて漸近操作を行いますが，次数を高くとると乗算回数が多くなります．そこで，次数は2次(乗算1回)として，以下のように漸化式を定義しましょう．

$$1-d_{i+1}=(1-d_i)^2 \quad \cdots\cdots(6)$$

式(6)を変形すると，

$$1-d_{i+1}=(1-d)^{2^i} \quad \cdots\cdots(7)$$

となり，サイクルが進むほど$1-d_i\to 0$を実現できることが確認されました．また，式(6)は次式のように変形できます．

$$d_{i+1}=d_i(2-d_i) \quad \cdots\cdots(8)$$

式(8)に式(3)を代入すると，式(9)が得られます．

$$x_i=2-d_i \quad \cdots\cdots(9)$$

これを順次d_i，z_iに乗算していくことで，$d_i\to 1$，$z_i\to q$が実現できます．

● 誤差があることを考慮して変数のビット幅を決める

演算回路(ハードウェア)における変数のビット幅をkとすると，d_iは$1-2^{-k}$以上には1に近づくことができません．したがって，$2^i\geqq k$を満たすiに至った時点で演算の反復動作は終了となります．このことから，反復回数iは，次の式で与えられます．

$$i\geqq \log_2 k \quad \text{を満たす最小の整数} \quad \cdots\cdots(10)$$

回復法や非回復法などの場合，除数，被除数，商，余の関係には誤差がありませんでした．一方，反復乗算法では「得られたq(商)は基本的に誤差を含んでいる」ということに注意しなくてはなりません．演算回路に誤差がなかったとしても，演算方法そのものに「d_iは$1-2^{-k}$以上には1に近づくことができない」という大前提があるので，分子のqもその分小さな値をとることになるからです．

また，乗算を行った結果，変数のビット幅は$2k$となります．これを次のサイクルにそのまま持ち込むと，変数の幅がサイクルを重ねるたびに大きくなってしまい，現実的ではありません．乗算結果はkビットに丸めて，次のサイクルに渡すのが一般的でしょう．この丸めによっても誤差が生じることになります．丸めによる誤差をεとすると，誤差を含んだd_i'は式(11)のように表現できます．

$$d_i'=d_i+\varepsilon \quad \cdots\cdots(11)$$

式(11)を用いて式(3)を書き直すと，式(12)が得られます．

$$\left.\begin{array}{l} d_{i+1}'=(d_i+\varepsilon)(2-d_i-\varepsilon) \\ \qquad =d_i(2-d_i)+2\varepsilon(1-d_i)+\varepsilon^2 \\ z_{i+1}'=z_i(2-d_i-\varepsilon) \\ \qquad =z_i(2-d_i)+\varepsilon z_i \end{array}\right\} \quad \cdots\cdots(12)$$

丸めを単純な切り捨てで行ったとすると，$\varepsilon<1$ulp(unit in least position；最下位ビット)となります．ε^2は十分小

さく無視できるものと考えると，誤差として支配的なのは式(12)の灰色の部分となります．この誤差はサイクルを重ねるたびに累積されるため，求める商のけたによって反復回数を決定した後，累積誤差の最大値を考慮して演算回路における変数のビット幅kを決める必要があります．

2 反復乗算法の回路化

図1に，反復乗算法による除算器のブロック図を，**リスト1**にそのVerilog HDLのRTL記述を示します．

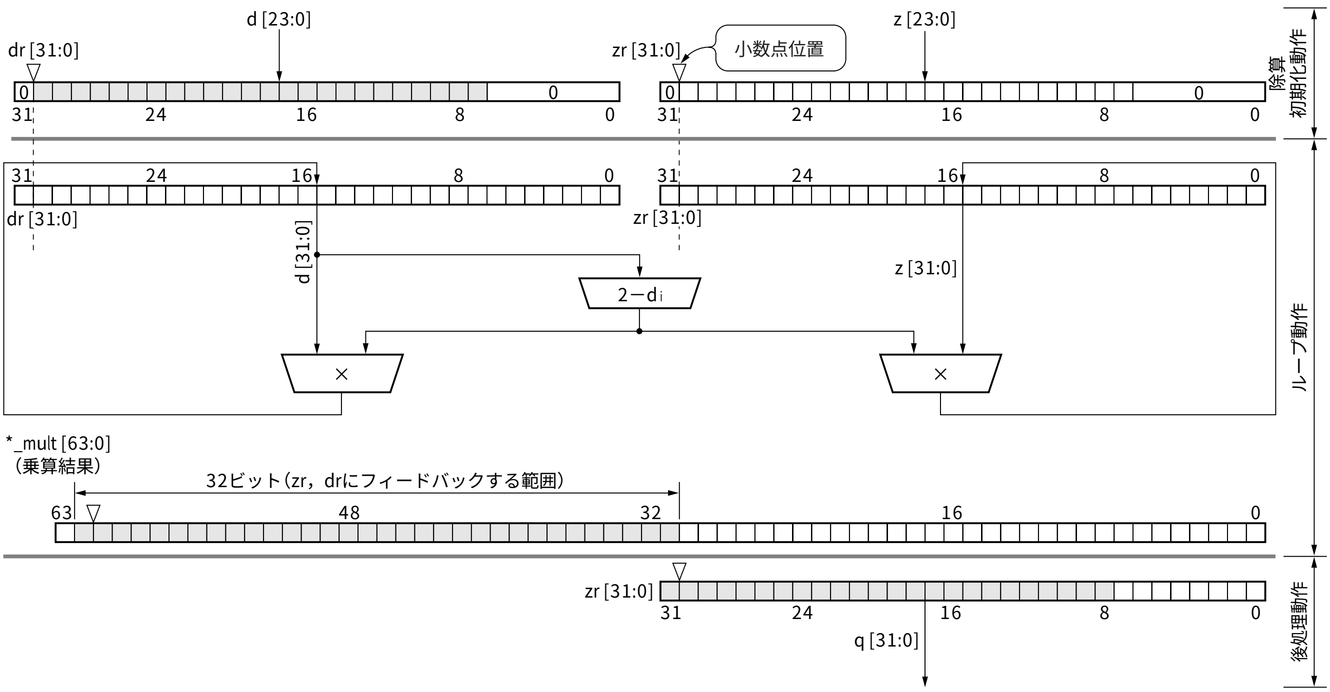


図1　反復乗算法による除算器ブロック図
図には，前処理，ループ(反復)，後処理とあるが，実際に演算処理を行っているのはループ部のみ．

リスト1　図1の回路構成のRTL記述(div_convergence_a.v)

```
module  div_convergence_a (
  q     , // quotient [23:00]:fraction
  busy  , // busy

  z     , // dividend [23:00]:fraction
  d     , // divisor  [23:00]:fraction
  start , // start (divider trigger)
  clk   , // clock
  re    ); // reset ena

// io
  input  [23:00]  z     ;
  input  [23:00]  d     ;
  input           start ;
  input           clk   ;
  input           re    ;

  output [23:00]  q     ;
  output          busy  ;

// reg
  reg    [02:00]  i     ;
  reg    [31:00]  zr    ; // [31]:int [30:00]:frac
  reg    [31:00]  dr    ; // [31]:int [30:00]:frac

// sequence (シーケンサ)
  wire  en_seq = ( i > 0 ) ;

  always @( posedge clk ) begin
       if      ( re )     i <= 3'h0     ;
       else if ( start )  i <= 3'h5     ;
       else if ( en_seq ) i <= i - 1'b1 ;
  end

// function (演算処理部)
  wire [31:00] d_cmpl = ~dr + 1'b1  ; // drの2の補数
  wire [63:00] d_mult = dr * d_cmpl ;
                                   // 分母にdrの2の補数を乗算
  wire [63:00] z_mult = zr * d_cmpl ;
                                   // 分子にdrの2の補数を乗算
  // next reg contents (次のサイクルのためのレジスタ内容)
  wire [31:00] dw     = d_mult[62:31] ; // 分母の丸め
  wire [31:00] zw     = z_mult[62:31] ; // 分子の丸め

  always @( posedge clk ) begin
       if      ( start )  zr <= { 1'b0,z,7'h00 } ;
       else if ( en_seq ) zr <= zw               ;
  end

  always @( posedge clk ) begin
       if      ( start )  dr <= { 1'b0,d,7'h00 } ;
       else if ( en_seq ) dr <= dw         ;
  end

  assign q    = zr[31:08] + zr[07] ; // 分子を丸めて出力
  assign busy = en_seq ;

endmodule
```

ここでは式(4)を満たすための入力の正規化は済んでいるものとします．浮動小数点演算の様相を呈しているので，zについても同じように正規化を施すこととし，単精度の演算を意識して入力のビット幅を24ビットとしました．式(10)より反復回数は5となります．反復回数を5とすると32ビットまで演算できるので，内部演算のビット幅は32ビットとしましたが，実際には28ビット程度で十分なはずです．

ブロック図は，前処理，ループ(反復)，後処理に分けて描いてありますが，前処理は24ビットの入力の32ビット・レジスタへの割り当てを，後処理は32ビット・レジスタから24ビットの商を取り出す位置を示しているだけです．すべて反復動作のみで除算が完了します．

● ある程度の精度であればFPGAのハード・マクロを利用

乗算器はターゲット・デバイスによって実現方法が異なるため，**リスト1**のRTL記述では単純に乗算を示す演算子のみで表現しています．また，乗算は1クロックで処理できることを前提としています．このままでもターゲット・デバイスによっては実用可能なはずですが，あくまでも演算方式を理解してもらうための参考記述とお考えください．シミュレーションで分母と分子の漸近動作を確認してもらえれば，と思います．

乗算反復法を実際に用いる場合，動作速度を決定するのはdの2の補数演算と乗算の部分になります．ここが設計者の腕の見せどころとなるでしょう．シーケンスのくふうで乗算器を1個で済ますのもおもしろいと思います．

図2に，シミュレーション結果の例を示します．$d = 0.5$, $z = 0.75$の場合，商は1.5になります．反復回数が大きくなるほど分母(d)は1に，分子(z)は1.5に近づいているのがわかります．

乗算反復法は乗算器を用いるということで，大げさな方式と思われがちですが，近ごろのFPGAのように乗算器がハード・マクロとして用意されているのであれば，比較的採用しやすいのではないかと筆者は考えます．もっとも，高精度なものが必要となると，乗算器も大きなものが必要になるので，適用範囲には限界があるのかもしれません．

3 Newton-Raphson法による逆数演算

本稿の最初に，漸近法のアルゴリズムの例としてNewton-Raphson法の名を挙げました．この方法は著名なのですが，除算に適用しようとするとめんどうです．除算は引き数が除数と被除数の二つですが，Newton-Raphson法は微分を使用するため，2変数ではぐあいが悪いのです(不可能ではないが，除算回路の実現という目的からは適当でないといえる)．

しかし，除算を「被除数と除数の逆数の乗算」ととらえれば，除数の逆数を求めるのにNewton-Raphson法は有効です．「逆数を求めて，さらに乗算するなんて二度手間だ」と思われるかもしれませんが，例えばアファイン変換による座標変換などでは共通の分母で複数の分子を除算するといった操作が行われます．このような場合，めんどうな除算は逆数として1回だけ求め，後はそれぞれの分子に乗算するという方法が有効です．

● x_iの接線とx軸の交点で漸化式を定義する

それでは，逆数演算にNewton-Raphson法を適用してみましょう．まずはアルゴリズムからです．**図3**にその原理を示します．ここで，式(13)に示す関数を定義します．

$$f(x) = 1/x - d \quad \cdots\cdots (13)$$

式(13)において，$f(x) = 0$とおくと$x = 1/d$を根に持ちます．言い換えれば，$x = 1/d$でx軸との交点を持ちます．このようすが**図3**に示されています．

さて，$x = x_i$における$y = f(x)$への接線を定義すると，**図3**における直線l_iになります．直線l_iとx軸の交点をx_{i+1}とすると，$x = x_{i+1}$における新たな接線l_{i+1}が定義され，これとx軸の新たな交点x_i+2が求められます．図からもわかるように，この操作を繰り返すことでx_iが求められるべ

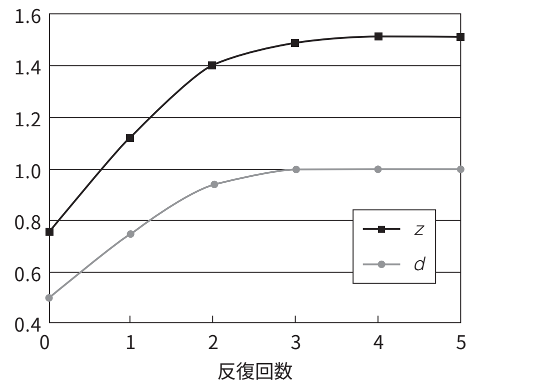


図2　乗算反復法による分母と分子の漸近
$d = 0.5$，$z = 0.75$の場合，商は1.5になる．反復回数が大きくなるほど分母(d)は1に，分子(z)は1.5に近づいていくのがわかる．

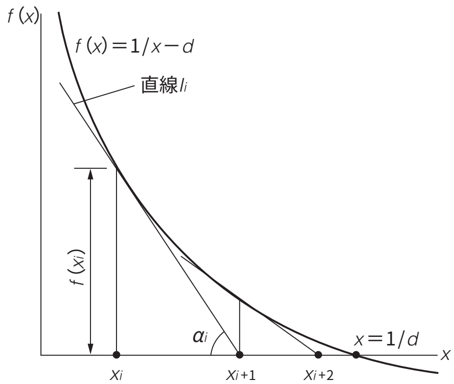


図3　Newton-Raphson法による逆数演算の原理
関数$f(x) = 1/x - d$は，$1/d$がx軸との交点となる．

き$x = 1/d$に近づいていきます．つまり，x_iとx_{i+1}の関係を定義する漸化式と，初期値x_0が定義できれば，漸近法として逆数演算が可能になります．このようにx_iにおける接線とx軸の交点で漸化式を定義するのがNewton-Raphson法です．

では，漸化式を求めることにしましょう．まず，$x = x_i$における$y = f(x)$への接線は，式(14)で定義されます．$f'(x_i)$は$x = x_i$における接線の傾きを導関数を用いて表現したものです．

$$y - f(x_i) = f'(x_i)(x - x_i) \quad \cdots\cdots (14)$$

この直線とx軸の交点x_i+1は式(14)において$y = 0$としたときの解なので，式(15)のように表せます．

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad \cdots\cdots (15)$$

ここで，

$$f'(x) = -1/x^2 \quad \cdots\cdots (16)$$

なので，式(15)に式(13)と式(16)を代入すると，次の漸化式が得られます．

$$x_{i+1} = x_i(2 - x_i \cdot d) \quad \cdots\cdots (17)$$

後は，初期値が定義できれば逆数演算が成立します．iステップ目の誤差を$\varepsilon_i = 1/d - x_i$とすると，

$$\begin{aligned}\varepsilon_{i+1} &= 1/d - x_{i+1} \\ &= 1/d - x_i(2 - x_i \cdot d) \\ &= d(1/d - x_i)^2 \\ &= d(\varepsilon_i)^2 \quad \cdots\cdots (18)\end{aligned}$$

ここで，反復乗算法と同じく，式(4)のように正規化しておくと$\varepsilon_{i+1} < (\varepsilon_i)^2$となります．式(4)に従って，

$$1 < x_0 \leqq 2 \quad \cdots\cdots (19)$$

となるように選択されていれば，漸近が保証されます．つまり，初期値はdの値に関係なく，例えば1.5の固定値としてもかまわないわけです．

しかし，式(18)から明らかなことですが，iステップ目における商の精度をa_iビットとすると，$i+1$ステップ目では$2a_i$ビット以上の精度となるので，x_0の選択はその後の反復回数を左右することになり，重要です．

例えば，4ビット精度のx_0で演算を開始すれば，3回の反復で32ビット精度まで追い込むことが可能となります．以上から，所望の商のビット精度をk，初期値のビット精度をa_0とすると，必要な反復回数iは次式で求められます．

$$i \geqq \log_2(k/a_0) \quad \text{を満たす最小の整数} \quad \cdots\cdots (20)$$

● テーブルを使って適切な初期値を与える

図4にNewton-Raphson法による逆数器のブロック図を，**リスト2**にそのVerilog HDLのRTL記述を示します．反復乗算法の場合と同じく，式(4)を満たすための入力の正規化は済んでいるものとし，入力の条件も24ビット幅としました(ブロック図およびRTL記述における注意点も反復乗算法の場合と同じ)．

すでに説明したとおり，Newton-Raphson法では初期値をより良い精度で与えることで反復回数を少なくできます．ここでは，1.5などの固定値ではなく，テーブルを用いて初期値を適切に与えることにしましょう．

しかし，テーブルがあまりにかさばりすぎるのも問題です．ここでは4ビット精度で初期値x_0を与えることにします．**リスト2**の`function`文にある`case`文がこのテーブルを表しています．$1/2 \leqq d < 1$の範囲を24ビット表現で入力するので，MSB(most significant bit)はつねに‘1’となります．したがって，$d[22:19]$でテーブルを索引すればよいことになります．本来，出力の範囲は式(19)のとおりですが，$1 < x_0 < 2$となるようにテーブルの内容を選択すれば，やはりMSBはつねに‘1’となります．このため，

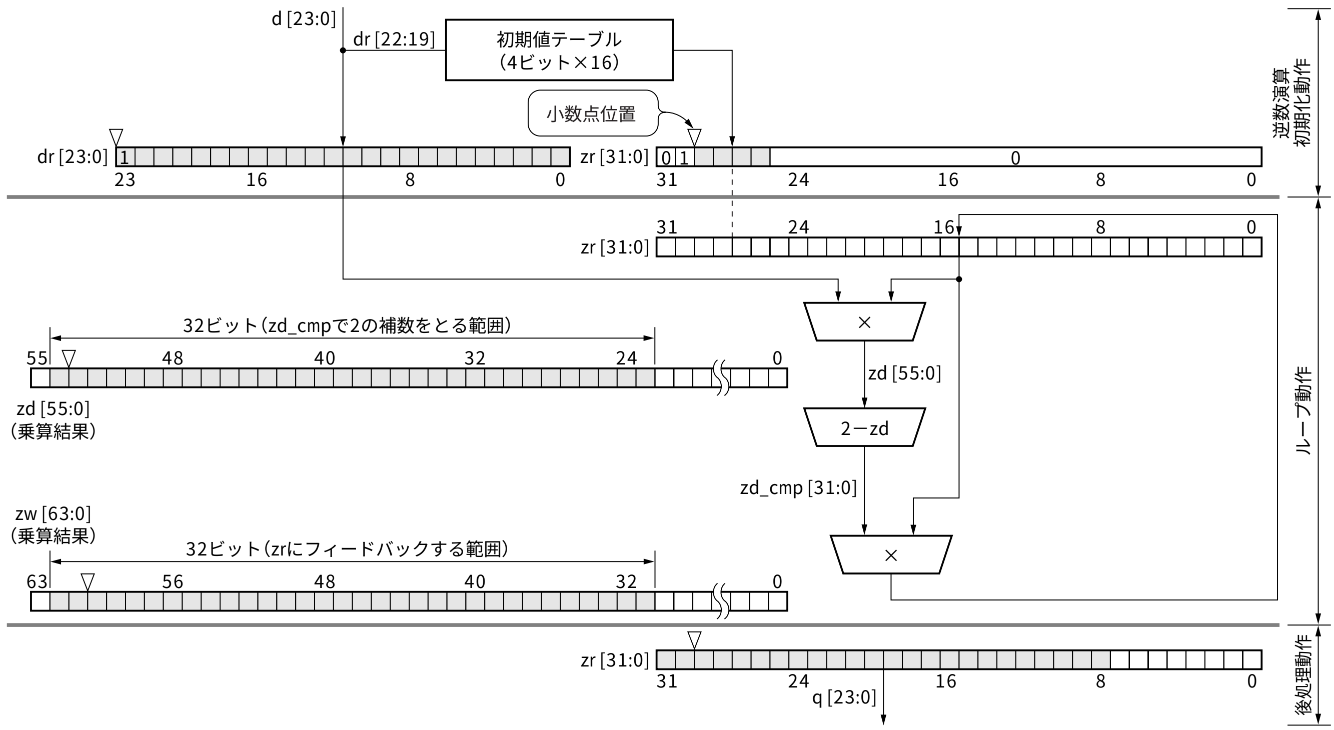


図4　Newton-Raphson 法による逆数器ブロック図

反復回数を少なくするため，ここでは初期値として固定値ではなく，テーブルを用いて適切な値を与えることにした．

リスト2　図5の回路構成のRTL記述(div_convergence_b.v)

```
module div_convergence_b (
  q     , // quotient [23:22]:int [21:00]:fraction
  busy  , // busy

  d     , // divisor  [23:00]:fraction
  start , // start (divider trigger)
  clk   , // clock
  re   ); // reset ena

// io
  input  [23:00] d     ;
  input          start ;
  input          clk   ;
  input          re    ;

  output [23:00] q     ;
  output         busy  ;

// reg
  reg    [02:00] i  ;
  reg    [31:00] zr ; // [31:30]:int [29:00]:frac
  reg    [23:00] dr ; //             [23:00]:frac

// sequence (シーケンサ)
  wire en_seq = ( i > 0 ) ;

  always @( posedge clk ) begin
        if      ( re )     i <= 3'h0     ;
        else if ( start )  i <= 3'h3     ;
        else if ( en_seq ) i <= i - 1'b1 ;
  end

// function (演算処理部)
  // 初期値テーブル
  function  [03:00] finix0 ;
        input [22:19] d_uppr ;
        case  ( d_uppr )
            4'h0 : finix0 = 4'hF ;
            4'h1 : finix0 = 4'hD ;
            4'h2 : finix0 = 4'hC ;
            4'h3 : finix0 = 4'hA ;
            4'h4 : finix0 = 4'h9 ;
            4'h5 : finix0 = 4'h8 ;
            4'h6 : finix0 = 4'h7 ;
            4'h7 : finix0 = 4'h6 ;
            4'h8 : finix0 = 4'h5 ;
            4'h9 : finix0 = 4'h4 ;
            4'ha : finix0 = 4'h3 ;
            4'hb : finix0 = 4'h3 ;
            4'hc : finix0 = 4'h2 ;
            4'hd : finix0 = 4'h1 ;
            4'he : finix0 = 4'h1 ;
            4'hf : finix0 = 4'h0 ;
        endcase
  endfunction
  wire [29:26] z0 = finix0 (d[22:19] ); // 初期値の決定

  // loop
  wire [55:00] zd     = zr * dr            ;
                    // xi*d        [55:54]:int [53:00]:frac
  wire [31:00] zd_cmp = ~zd[54:23] + 1'b1 ;
                    // 2-xi*d          [31]:int [30:00]:frac
  wire [63:00] zw     = zr * zd_cmp       ;
                    // xi*(2-xi*d) [63:61]:int [60:00]:frac

  always @( posedge clk ) begin
        if    ( start )  zr <= { 2'h1,z0,26'h0000000 };
        else if ( en_seq ) zr <= zw[62:31] ;
  end

  always @( posedge clk ) begin
        if      ( start )  dr <= d ;
  end

  assign q    = zr[31:08] + zr[07] ;
  assign busy = en_seq ;

endmodule
```

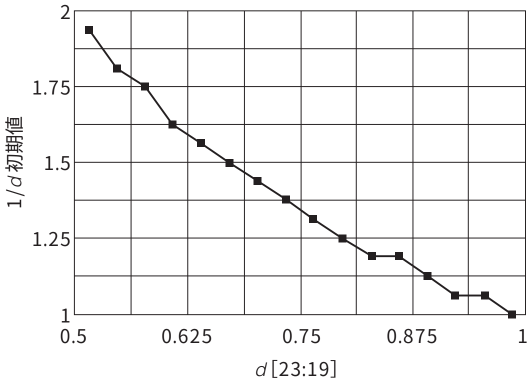


図5　逆数演算初期値テーブル

4ビット幅で構成した場合のMSBを含むテーブルの内容を示す．なお，このテーブルの内容だと，4ビット精度には若干不足する場合があるのだが，実使用時に問題がないのでこのままとしてある．

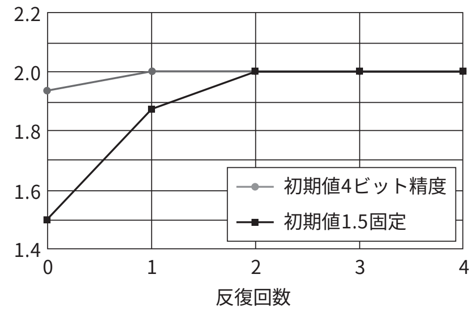


図6　初期値の違いによる漸近のようす

24ビット精度の商を求めているので図からはわかりにくいが，初期値を固定の1.5とすると4ビット精度のテーブルを用いた場合よりも反復回数が一つ多くなり，4回となる．

この部分はテーブルに含まずレジスタの初期化で与えることとし，4ビット幅でテーブルの内容を構成します．MSBを含んだテーブルの内容は**図5**のようになります．

以上の条件から，式(20)により反復回数は3となります．また，**図6**に初期値を変化させた場合の漸近のようすを示します．

＊　＊　＊

今回は，漸近法に焦点をあててみました．設計者として駆け出しのころ，筆者にとってもこのような乗算の手法はとっつきにくく，できるだけ避けるようにしていた記憶があります．しかし，数式に惑わされず，ツボさえ押さえておけばめんどうなものではないことが理解できたことと思います．精度を確保する反復回数も，意外と単純な計算で求められます．構造的な面で特殊なので，オールマイティとは言いがたいのですが，条件が合えば使ってみるのもおもしろいでしょう．

すずき・しょうじ
スタンダード・ロジック(株)

<筆者プロフィール>
鈴木昌治．10月初旬に，「グランフォンド糸魚川」(120kmほどの山あり谷ありコースを走破する自転車遠足)に参加．ひどい雨の中での走行となりましたが，参加賞としてカニをもらってホクホク．